Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

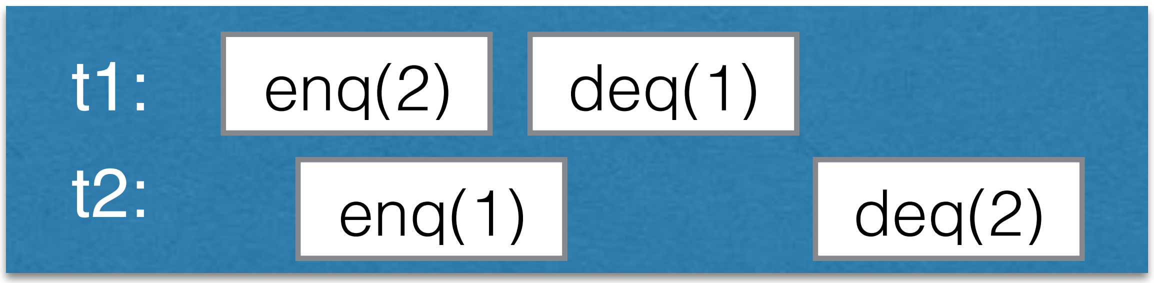
joint work with:

Andreas Haas Google
Andreas Holzer UNIVERSITY OF TORONTO
Michael Lippautz Google
Ali Sezgin UNIVERSITY OF CAMBRIDGE
Tom Henzinger IST AUSTRIA
Christoph Kirsch UNIVERSITY of SALZBURG
Hannes Payer Google
Helmut Veith TU WIEN

Helmut Veith Memorial Workshop, Obergurgl, 1.2.17

Concurrent Data Structures
Correctness and Performance

Semantics of concurrent data structures



e.g. pools, queues, stacks

- Sequential specification = set of legal sequences

e.g. queue legal sequence
enq(1)enq(2)deq(1)deq(2)

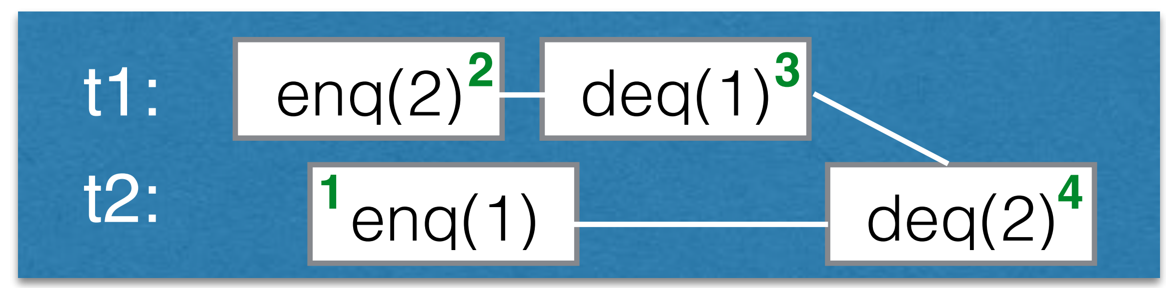
- Consistency condition = e.g. linearizability / sequential consistency

e.g. the concurrent history above is a linearizable queue concurrent history

Consistency conditions

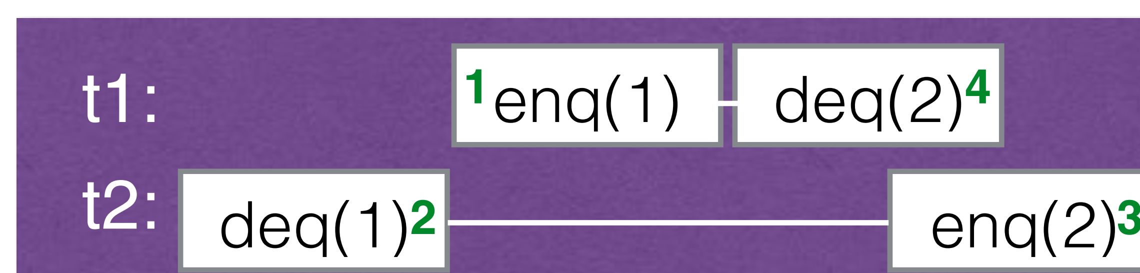
there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]

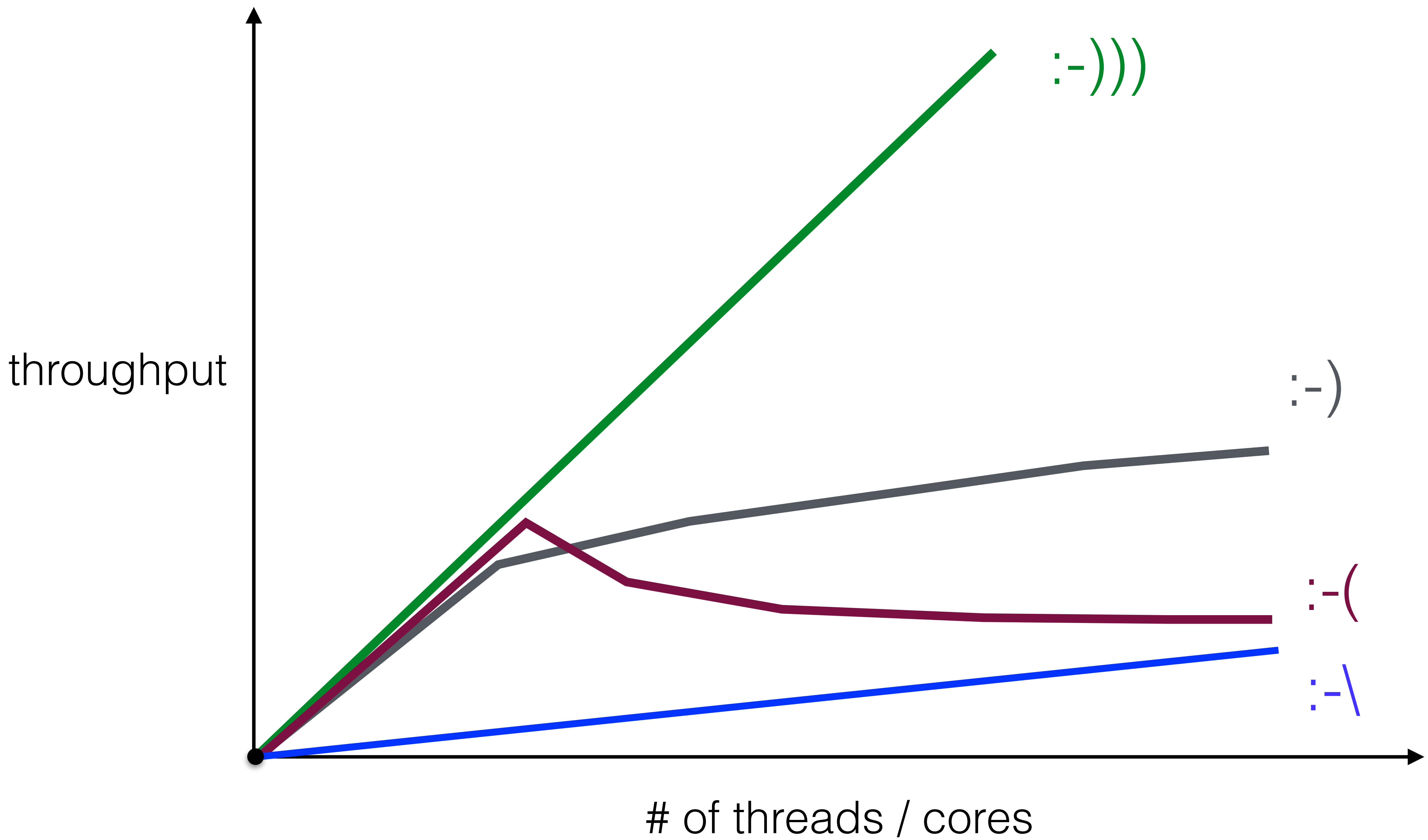


Sequential Consistency [Lamport'79]

there exists a legal sequence that preserves per-thread precedence (program order)



Performance and scalability



Relaxations allow trading

correctness
for
performance

provide the potential for better-performing implementations

Relaxing the Semantics

not "sequentially correct"

Quantitative relaxations - POPL13
Henzinger, Kirsch, Payer, Sezgin,S.

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Local linearizability - CONCUR16
in this talk

for queues only
(feel free to ask for more)

too weak

Local Linearizability main idea

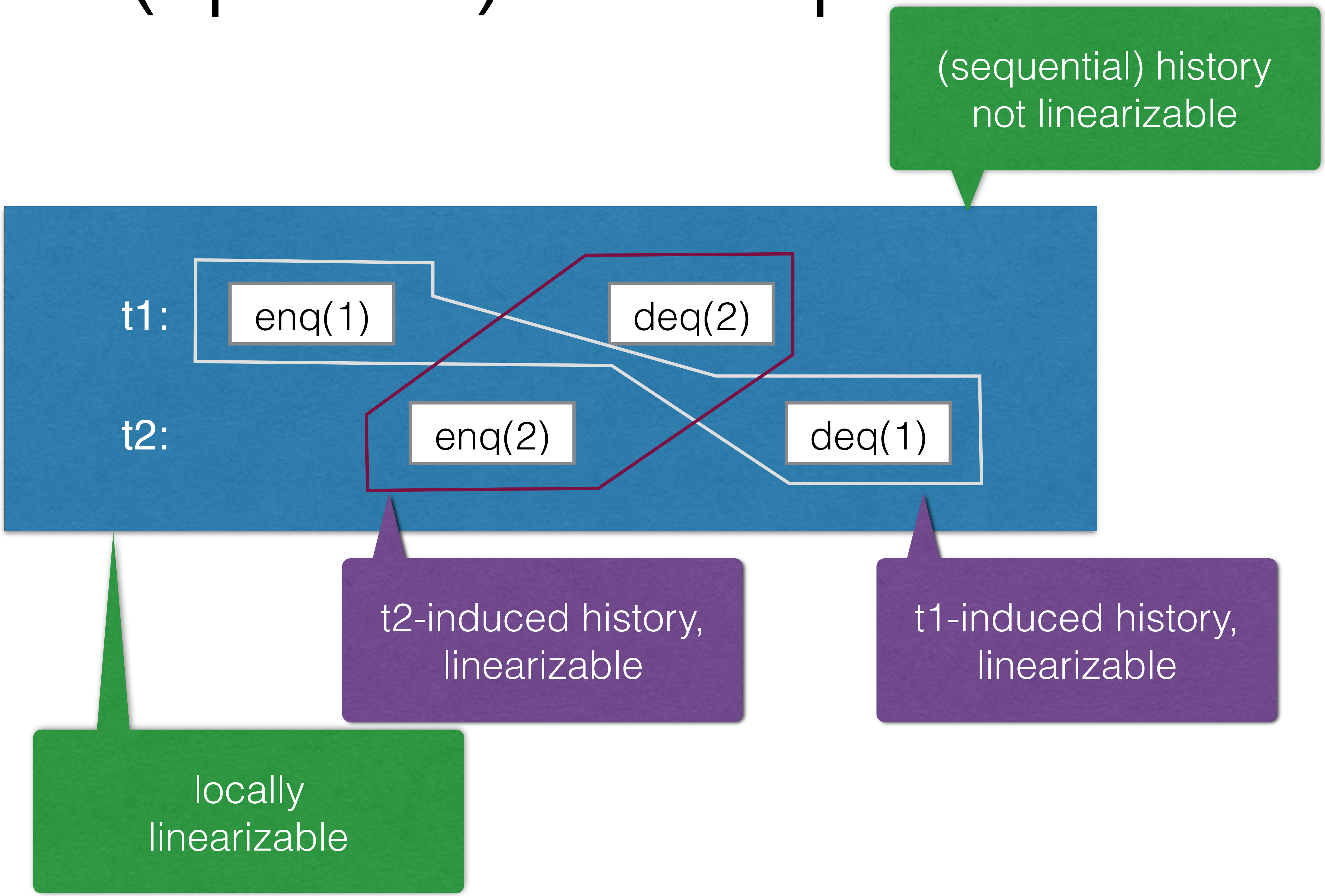
Already present in some shared-memory consistency conditions (not in our form of choice)

- Partition a history into a set of local histories
- Require linearizability per local history

no global witness

Local sequential consistency… is also possible

Local Linearizability (queue) example



Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

For a history **h** with a thread T, we put

$$I_T = \{enq(x)^T \in \mathbf{h} \mid x \in V\}$$

$$O_T = \{deq(x)^{T'} \in \mathbf{h} \mid enq(x)^T \in I_T\} \cup \{deq(empty)\}$$

in-methods of thread T are enqueues performed by thread T

out-methods of thread T are dequeues (performed by any thread) corresponding to enqueues that are in-methods

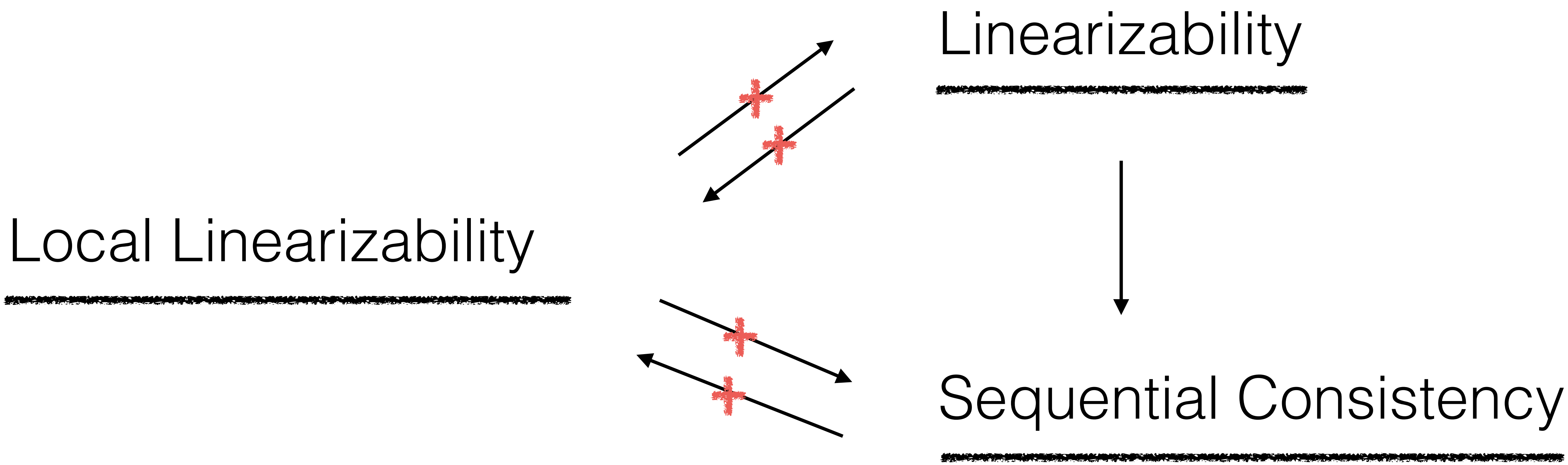
h is locally linearizable iff every thread-induced history

$$\mathbf{h}_T = \mathbf{h} \mid (I_T \cup O_T)$$

is linearizable.

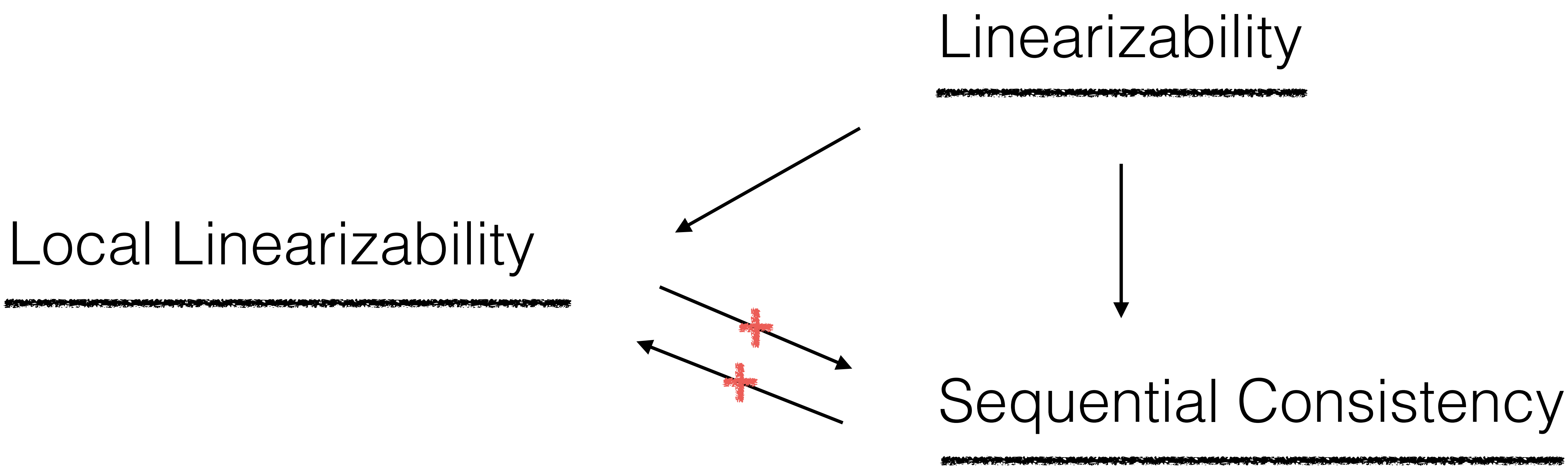
Where do we stand?

In general



Where do we stand?

For queues (and most container-type data structures)



Properties

Local linearizability is compositional

> like linearizability
> unlike sequential consistency

h (over multiple objects) is locally linearizable
iff
each per-object subhistory of **h** is locally linearizable

Local linearizability is modular / "decompositional"

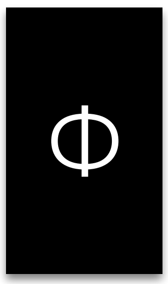
> uses decomposition into smaller histories, by definition

↓

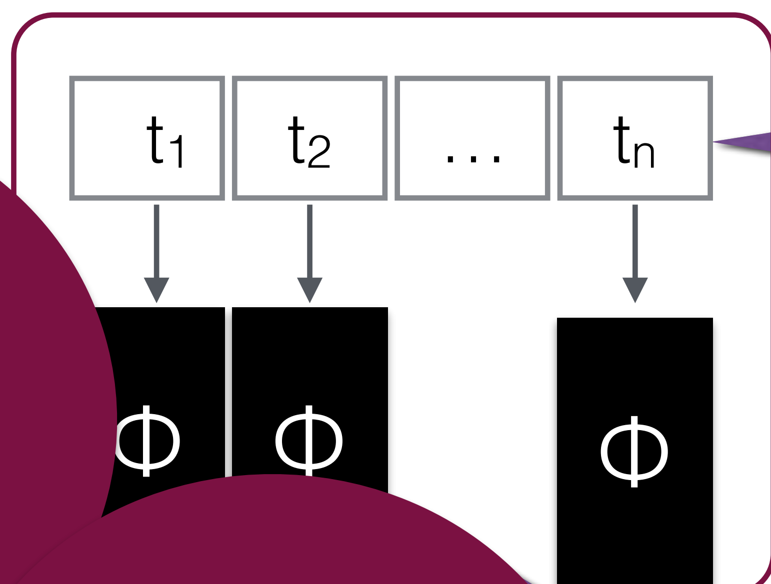
may allow for modular verification

Generic Implementations

Your favorite linearizable data structure implementation



turns into a locally linearizable implementation by:



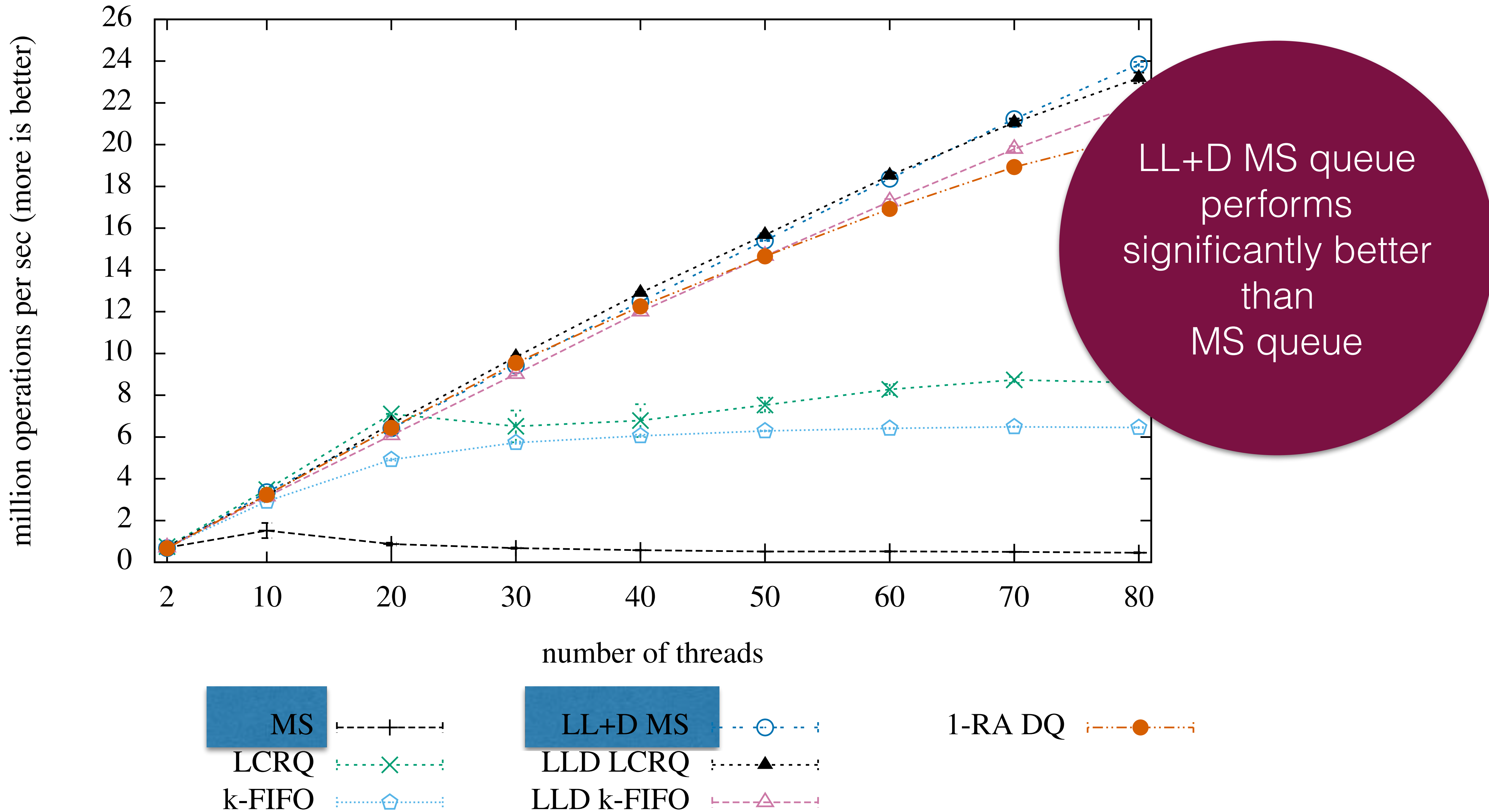
segment of possibly dynamic size (n)

LLD Φ (locally linearizable)

LL+D Φ (also pool linearizable)

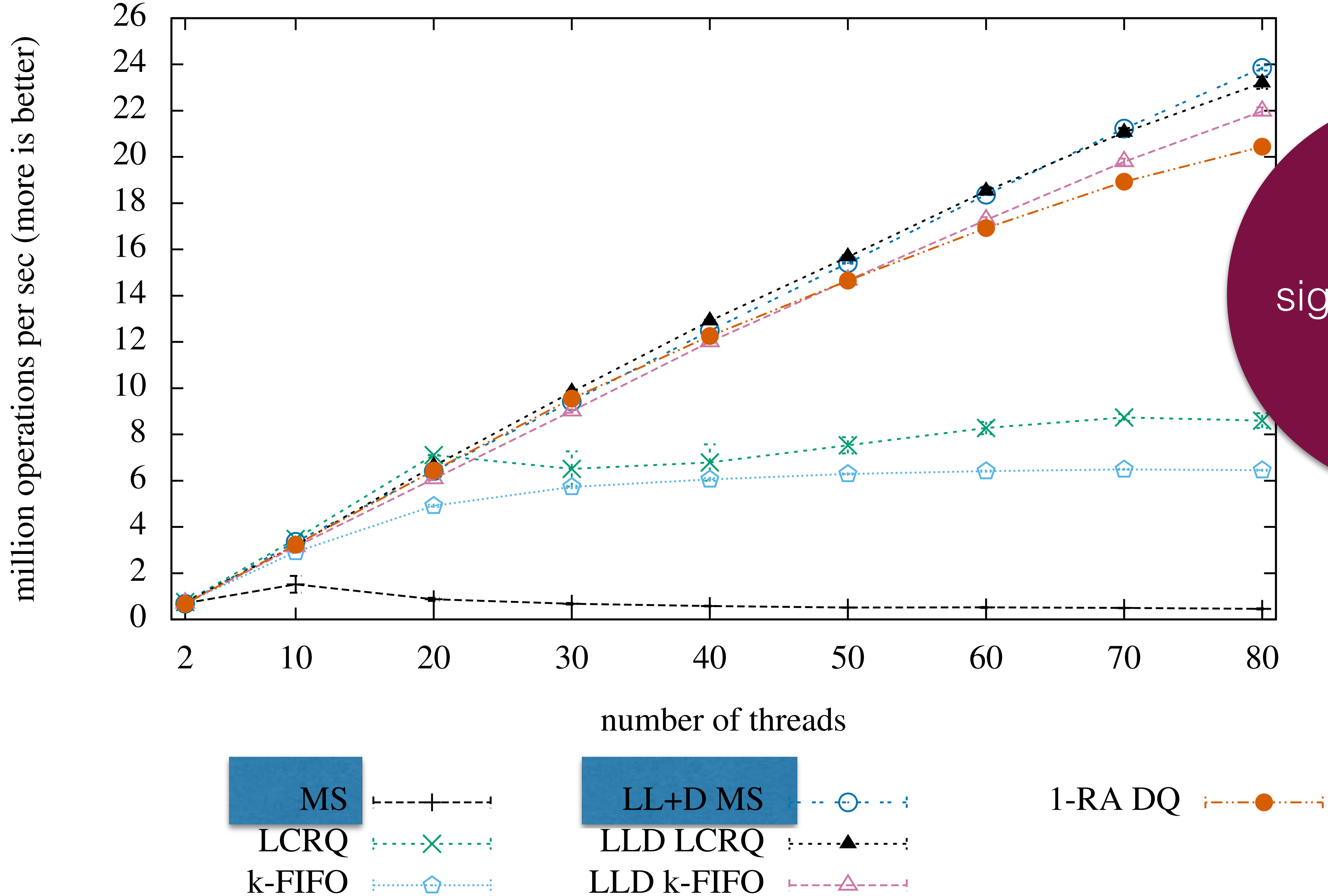
local inserts / global (randomly distributed) removes

Performance



(a) Queues, LL queues, and "queue-like" pools

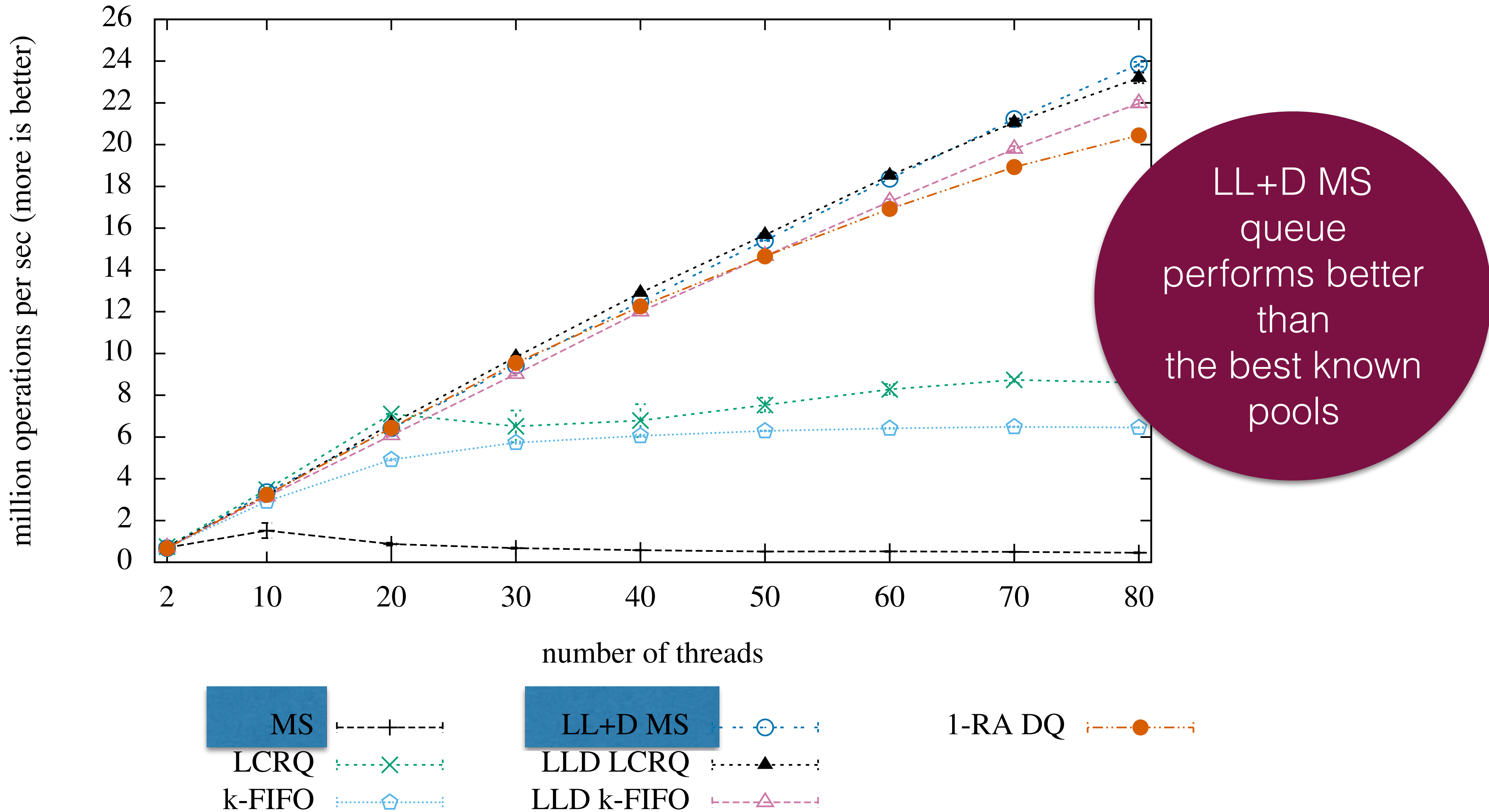
Performance



(a) Queues, LL queues, and "queue-like" pools

LLD Φ performs significantly better than Φ

Performance



(a) Queues, LL queues, and “queue-like” pools

Thank You!

Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

joint work with:

Andreas Haas Google

Andreas Holzer UNIVERSITY OF TORONTO

Michael Lippautz Google

Ali Sezgin UNIVERSITY OF CAMBRIDGE

Tom Henzinger IST AUSTRIA

Christoph Kirsch UNIVERSITY of SALZBURG

Hannes Payer Google

Helmut Veith TU WIEN